Software Architecture of Sensor Data Distribution In Planetary Exploration

Charles Lee
SAIC
NASA Ames Research Center
Moffett Field, CA. 94035
650-604-6054
clee@mail.arc.nasa.gov

Richard L. Alena
NASA Ames Research Center
Moffett Field, CA. 94035
650-604-0262
Richard.L.Alena@nasa.gov

Thom Stone
Computer Science Corporation
NASA Ames Research Center
Moffett Field, CA. 94035
650-604-4971
tstone@arc.nasa.gov

John Ossenfort
SAIC
NASA Ames Research Center
Moffett Field, CA. 94035
650-604-0159
jossenfort@mail.arc.nasa.gov

Ed Walker
MCT
NASA Ames Research Center
Moffett Field, CA. 94035
650-604-1086
ctmwalker@mail.arc.nasa.gov

Hugo Notario
Foothill-DeAnza College
NASA Ames Research Center
Moffett Field, CA. 94035
650-604-4036
hnotario@mail.arc.nasa.gov

Abstract—Data from mobile and stationary sensors[1,2] will be vital in planetary surface exploration. The distribution and collection of sensor data in an ad-hoc wireless network presents unique challenges. Some of the conditions encountered in the field include: irregular terrain, mobile nodes, routing loops from clients associating with the wrong access point or repeater, network routing reconfigurations caused by moving repeaters, signal fade, and hardware failures. These conditions present the following problems: data errors, out of sequence packets, duplicate packets, and drop out periods (when the node is not connected). To mitigate the effects of these impairments, robust and reliable software architecture tolerant of communications outages must be implemented. This paper describes such a robust and reliable software infrastructure that meets the challenges of a distributed ad hoc network in a difficult environment and presents the results of actual field experiments testing the principles and exploring the underlying technology.

TABLE OF CONTENTS

[1] 0-7803-9546-8/06/$20.00© 2006 IEEE.
[2] IEEEAC paper #1168, Version 1, Updated Oct 18, 2005

1. INTRODUCTION

Sensor data is an essential part of planetary surface exploration. Sensor data arises from status monitoring, system health assessment as well as scientific sampling. All are required for the success of the mission and sensor data is required for all aspects of a mission:

- Planning requires sensor data as input to determine location, environment and distances.
- Scheduling requires sensor information for calculating duration, time, position, and routes.
- Operation requires sensor information to calculate location, progress, health status, etc.

Sensor data steams are in real-time and are time critical. Parallel processing of sensor data to produce useful information introduces reliability issues. The two major causes of data loss are the burden of communications overhead and packet drops plus the difficulty of multithreaded programming. Packet loss in wireless systems can be caused by many factors:

- Congestion
- Moving out of range of base station
- RF interference
- Multi-paths
- Obstacles to line of sight
- Routing problems

The basic problem we are attempting to address is how to transmit sensor data accurately over a wireless infrastructure to single or multiple receivers where the networks may have short duration outages. In planetary exploration, environment could be very difficult where packet loss and even loss of signal, temporarily disconnected networks, specifically, short duration outages are the norm.

The solution chosen by us was to use software middleware techniques to overcome the communication blockage problems. Although this work was done for surface communications, it could well have application to other data management areas, such as orbital asset management, spacecraft tracking and control, and test bed data distributions. Satellite constellations usually consist of many instruments and sensors producing multiple steams to multiple consumers of the data. Satellites can experience periods of a high rate of data errors and periods of loss of signal. Like our surface data, spacecraft sensor data can have highly distributed users, some on different planets.

We experimented with different schema frameworks to determine a method to find an optimal system for robustness and reliability of sensor data for surface communications.

Ultimately, we selected Message Oriented Middleware (MOM) for our distributed infrastructure [3]. Message Oriented Middleware is a category of inter-application communication software that presents an asynchronous message-passing model as opposed to a request/response model.

MOMs have the following attributes:

- Fast
- Reliable
- Asynchronous
- Guaranteed message delivery
- Receipt notification
- Transaction control

As far as the client software is concerned, MOM is indistinguishable from real-time processing [4]. The primary advantage of a message-oriented communications protocol is the ability to store, route, and resend a message that needs to be delivered.

2. TECHNICAL BACKGROUND

Most MOM systems provide persistent storage to hold messages until they are successfully transferred. This means that it is not necessary for the sender and receiver to be connected when data are created. This is useful for dealing with faulty connections, unreliable networks, and timed connections (where communications is only available during predictable periods). It also means that if a receiver fails to receive a message for any reason, the sender can continue unaffected, since the messages will be held in the message store and will be transmitted when the receiver reconnects.

MOM systems usually present two messaging models, but not all MOMs support both models:

- Point-To-Point
- Publish-Subscribe

Point-to-point:

This model [2] is based on message stores known as queues. A sender sends a message to a specified queue and a receiver receives messages from the queue. A queue can have multiple senders and receivers, but an individual message can only be delivered to one receiver. If multiple receivers are listening for messages on a queue, the underlying MOM system usually determines which receiver will receive the next message. If no receivers are listening on the queue, messages remain in the queue until a receiver attaches to the queue.

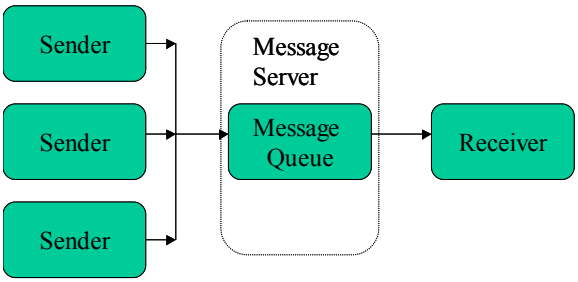


Figure 1 – Point to Point Messaging

Point-to-Point Messaging—the critical aspect of point-to-point messaging is that even though there may be multiple senders of messages, there is only a single receiver for the messages.

End (client) systems may only send messages, only receive messages, or both send and receive messages. Another client can be sending and/or receiving messages at the same time. In the simplest case, one client is the sender of the message and the other client is the receiver of the message.

There are two basic types of point-to-point messaging systems. The first one involves a client that directly sends a message to another client. The second and more common implementation is based on the concept of a Message Queue. Such a system is shown below.

Publish-Subscribe:

This model [1] is based on message stores known as topics. Publishers send messages to a topic. Multiple subscribers can retrieve messages from a topic. When multiple applications need to receive the same messages, Publish-Subscribe Messaging is required. The central concept in a Publish-Subscribe Messaging system is the "Topic." Multiple publishers may send messages to a topic, and all subscribers to that topic receive all the messages sent to that topic. This model, as shown in Figure 2, is extremely useful when a group of applications want to notify each other of a particular occurrence.

In Publish-Subscribe Messaging, there may be multiple senders and multiple receivers. Point-to-point can have multiple senders but only one receiver.

Overview of Message Services

Typically a message service is implemented using a Java framework. A Message-Driven Bean (MDB) is an Enterprise Java Bean (EJB) that functions as a message consumer. Unlike session beans or entity beans, clients cannot access MDBs directly. Also, unlike session beans and entity beans, an MDB does not have remote or home interfaces. The only access a client has to an MDB is through a JAVA Messaging Service (JMS) destination (topic or queue) to which the MDB is listening.

An MDB must implement two interfaces:

1. ***javax.jms.MessageListener***—this interface defines the *onMessage* callback method. When a message is put on the queue/topic, the *onMessage* method of the MDB is called by the EJB container and passes the actual message.

2. ***javax.ejb.MessageDrivenBean***—this is the EJB interface that contains the EJB lifecycle methods:
   - ***ejbCreate()***—called by the EJB container when the MDB is created.
   - ***ejbRemove()***—called by the EJB container when the MDB is destroyed or removed from the EJB pool.
   - ***setMessageDrivenContext(MessageDrivenContext context)***—called prior to *ejbCreate* and passed the message-driven context by the EJB container.

An MDB must declare deployment information about itself in a deployment-descriptor file named *ejb-jar-xml*. The EJB container handles the duties of subscribing the bean to the topic or connecting it to the queue based on information placed in the deployment descriptor. The context has runtime information, such as the actual transaction data.

The diagram in Figure 2 illustrates the interactions between a JMS message, a client, a topic, an application server, an EJB container, and MDB instances.

As mentioned before, MDBs do not have remote or local interfaces as with session beans and entity beans. MDBs are not located by client classes and client classes do not directly invoke methods on them. All access to a MDB is through a JMS topic or queue that directs messages at the MDB through the EJB container. The EJB container ultimately passes the JMS message to the MDB through the bean's *onMessage* method. All MDBs must implement the *javax.ejb.MessageDrivenBean* and *javax.jms.MessageListener* interfaces, as the example illustrates.

The JMS provides a standard Java-based interface to the message services of a MOM of a non-JAVA based schema. This means that various brands of middleware can interoperate.

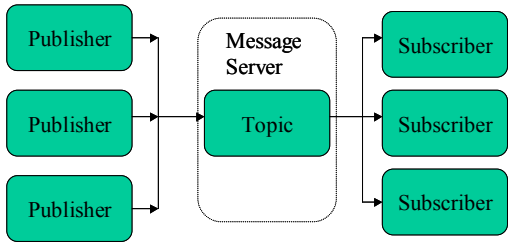


Figure 2 - Publish Subscriber Messaging

To illustrate the relationship of the classes, Figure 3 shows class hierarchy of the messaging services. It also shows the differences between the Publish-subscribe and Point-to-point messaging in table format.

JMS Parent	Publish-subscribe Domain	Point-to-point Domain
Destination	Topic	Queue
Connection Factory	Topic Connection Factory	Queue Connection Factory
Connection	Topic Connection	Queue Connection
Session	Topic Session	Queue Session
Message Producer	Topic Publisher	Queue Sender
Message Consumer	Topic Subscriber	Queue Receiver, Queue Browser

Figure 3 - Publisher and subscriber class hierarchy.

3. ARCHITECTURE DESIGN

We selected the Publish-Subscribe architecture for data distribution. In our project requirements, the data is to be distributed to multiple remote clients and the publisher may publish the data to a remote machine. Implementing the Publish-Subscribe architecture satisfies these requirements.

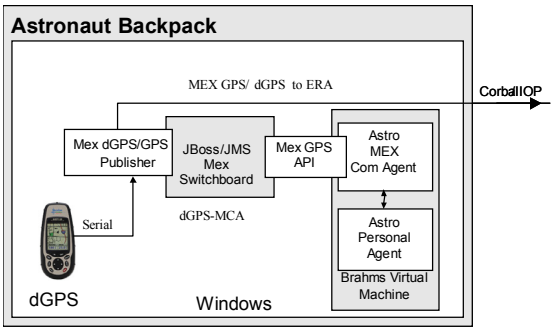


Figure 4 – GPS data distribution architecture.

During three years of field trials our software was deployed as follows.

Figure 4 shows an Astronaut carrying a backpack with our software running on a computer inside. A Global Positioning System (GPS) unit is connected to the computer and the data is distributed to the JMS server using the GPS server model. The client accesses the data by subscribing to the topic using the API provided by the GPS server developer. Biological information from sensors attached to the Astronaut is also distributed using this architecture as shown in Figure 5 below.

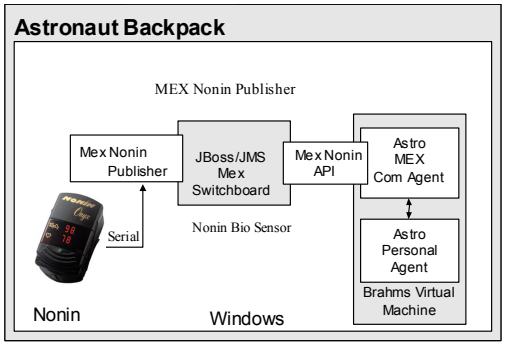


Figure 5 - Biosensor architecture.

We have collaborated with a Robotic Rover team (Extra Vehicular Activity Robotic Assistant [ERA]) from Johnson Space Center in several exploration field tests (see Lessons Learned). The sensor data needs to be distributed to the ERA server so that the robot can perform commands such as:

- Follow the astronaut
- Take a picture of an area that is of interest to the astronaut
- Take a picture of an astronaut
- Make a voice note
- Open a sample bag for a specimen

Since the ERA team is using a Common Object Request Broker Architecture (CORBA) framework [5,6] for their distributed object model, we needed to distribute the data across a CORBA object by connecting our CORBA client with the sensor and pushing the data to the Rover object running on a CORBA Object Request Broker (ORB).

The architecture of the data distribution to the ERA server is shown in Figure 6. The ERA implements a server (Executor) to accept the data and store it in local memory for a finite period of time. The data must be pushed at a rate that refreshes the data before the memory times out.

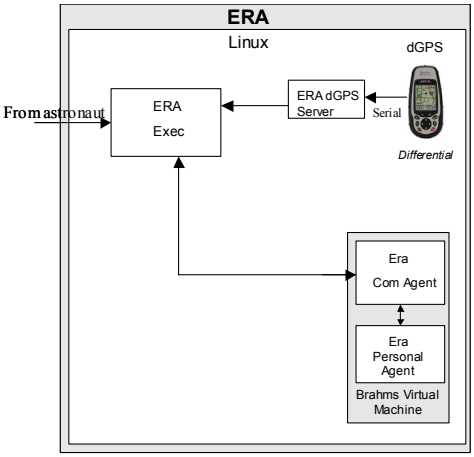


Figure 6 - The ERA CORBA server.

The subscribed client will receive the stream of data by intercepting the message listener. An example of this type of client is the rover monitor, which can show the movement of the rover on a map in the real time. Figure 7 shows the monitor screen from a pre-field test at Moffett Field in California.

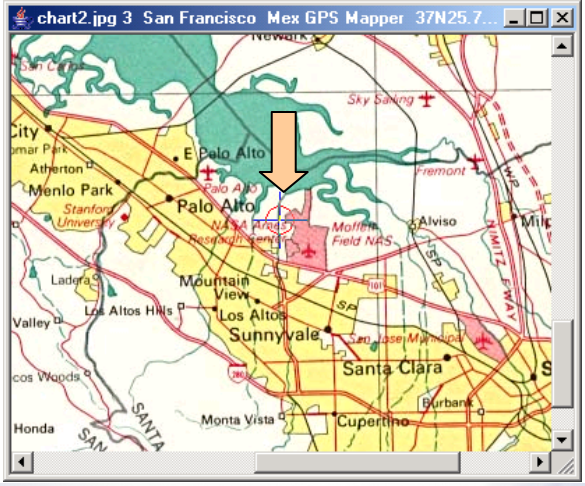


Figure 7 - Navigation monitor.

The circle with cross is the moving cursor that shows the rover location by interpreting the coordinates received from subscribing to the GPS topic. The accuracy of the location was within five centimeter in the last field test.

Reliable and Robust

The architecture we have described has the capability of providing reliability and robustness during short outages. However, some issues are not addressed directly by our architecture itself. Of most concern, are longer duration network outages in severe environments.

The low power output and delay sensitive protocols of 802.11b are prone to fading, especially when used over multi-hop long haul point-to-point circuits. Additional problems are caused when equipment moves out of the line of sight or encounters routing difficulties as equipment moves and long duration outages occur.

The first software challenge is to recognize that a communications outage is occurring. Fortunately, both JAVA and CORBA provide this functionality. JAVA automatically sets a software exception that can be detected and CORBA has a ping function that can be used to explicitly insure the network is connected before do the actually connection.

Figure 8 – Astronaut and robotic assistant at the MDRS in Utah.

We have the tested our methodology during simulation tests at the Mars Desert Research Station (MDRS) in Utah during the spring of each year from 2002-05. These "Mobile Agent" expeditions tested interactions between astronauts and robotic assistants. They were a collaborative effort between several NASA centers and university investigators. We used 802.11b to communicate between astronaut, robot and the base station. Some of these links were over several kilometers, which required installation of repeaters in temporary locations that subjected them to wind and rain. A satellite link was used to send the data via the NASA Research and Engineering Network (NREN) from base camp to multiple researchers at their home institutions.

As the astronauts and robots move, they come in and out of wireless signal coverage. Under such conditions, with short-term outages the norm, data distribution becomes unreliable. Connections can be lost either between the astronaut and the JMS server, or from the ERA robot to the JMS server (or both). Any of these data interruptions will keep the sensor data from reaching the proper destination.

A software workaround was devised to mitigate this impairment. A retry loop was established which continued to test the path until it has recovered. To prevent the retries from impacting data collection from the sensor, tying up CPU usage, or other resources, a counter was implemented

to wait a period of time (in seconds) before the connection was retried. The data was stored until connectivity was re-established.

The logic path used for implementing timeout loops in the subscriber model is as follows:

```
In the processing of data loop
If (reconnectCounter==0)
        DoReconnect();
        Publish();
Else
        ReconnectCounter--;
Endif

When (ConnectionException)
        SetReconnectCounter;
 End loop
```

Another important issue was when the SerialConnection class was used to acquire data from the COM port; it should not be interrupted by other tasks, as the data flow received was a continuous real-time stream. Separate threads were implemented to prevent interruption of data distribution and the SerialConnection class dedicated to acquiring and storing the data in memory for further processing.

4. Field Tests and Lessons Learned

Field tests took place at MDRS, in an isolated area in Utah. A satellite link was connected to the high-speed NREN backbone through Glenn Research Center (GRC) in Ohio. Astronauts (fully suited) were paired with robotic assistants. They communicated with each other over wireless links and the robot responded to voice commands from the astronaut. The robot contained a mobile wireless local area network (WLAN) repeater. The activities were monitored from a base camp several kilometers away. Additional repeaters were situated on all terrain vehicles (ATV) and nearby hilltops.

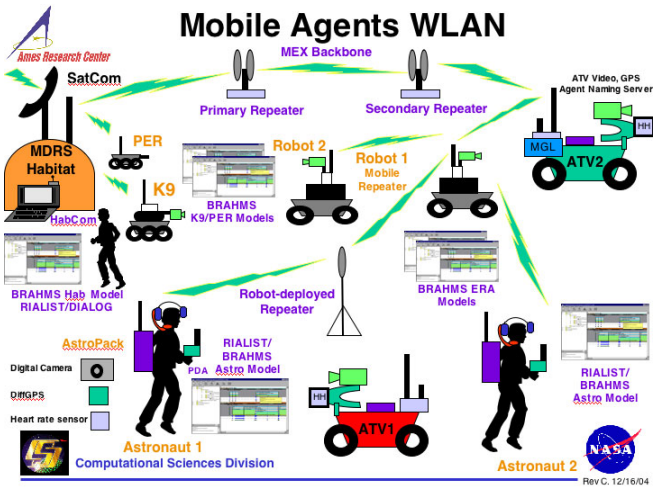


Figure 9 – Mobile Agents communication topology.

Experiences from the 2003 Mobile Agents field season included an attempt at Ames Research Center (ARC) to integrate the equipment (robots, wireless, etc.) and software from the many contributing groups at ARC and Johnson Space Center before deploying to the field, but lack of time and travel resources forced a very superficial integration effort. Various groups were also in development stage until just before the deployment. It was almost expected that we would experience problems and would have to redesign in the field.

The first week of the two-week period was marked by problems related to the substantial packet delay and frequent connectivity drops. The software responsible for GPS location service failed to correctly establish coordinates when the GPS real-time data from the equipment in the backpack was delayed. This made testing astronaut-robot-operations center voice recognition and command processing impossible. Additionally, overheating problems with the backpack computers and routing problems on our multi-hop wireless system were experienced.

Several steps to mitigate these problems were attempted. A Network Time Protocol (NTP [RFC 1305]) Timeserver was implemented to timestamp all GPS and biosensor data. This made it possible to correctly correlate the location of the astronaut or the robot with a time series. Sensor message processing was moved to a computer that had less network traffic and less extraneous processing. Also, publish-subscribe middleware was fully implemented.

These measures and further tuning of the wireless infrastructure to fix some routing problems, and adding an additional fan to the backpack led to several successful simulations.

Experiences during the 2004 field season were more positive, even with some rather severe weather and dust storms. Before deployment the astronaut backpack was redesigned, adding better ventilation to accommodate an updated rugged laptop computer, which had more memory and a mobile Pentium processor with a wider operating temperature range.

The WLAN complexity (mobile access points on ATV and mobile robots with repeater sites on far away hills) was simplified and multi-pathing and channel overlap were reduced. Routing between the elements was rationalized to prevent loops, which led to higher bandwidth and better throughput.

The distributed sensor architecture was tested on by moving the JMS server to various computers on the system. All worked as designed, although response was still slow but tolerable.

The 2005 season built on the successes of 2004. The software was optimized and the client API redesigned. The client is the "subscriber" to the sensor data message server. One improvement was to have the software "sleep" when no messages are in the queues and awake when they are available. This cut the CPU and memory requirements substantially. The sensor data have duty cycles of less than 10 hertz, thus leaving substantial periods for other software tasks. Improvements were also made in the voice loop software, the mobile agent software, and the voice recognition software. These upgrades produced improved performance and seamless transition in and out of wireless coverage.

5. CONCLUSIONS

The field tests and experiments show that the distributed components model utilizing the JMS architecture is very suitable for real time sensor data distribution. It produces reliable and robust data streams to multiple clients in real-time. The publish-subscriber model is very scalable, even for processing data from many sensors. For publishing data from multiple sensors, message beans and topics can be easily created for each occurrence of a sensor.

Longer duration network outages, which are common in the field, can be easily mitigated by simple software modifications.

These techniques have relevance to situations where multiple assets are distributed on the ground and in orbit, and sensor and other data are to be distributed to multiple consumers locally or on Earth.

REFERENCES

[1] Jameela Al-Jaroodi, Nader Mohamed, Hong Jiang, and David Swanson, "Middleware Infrastructure for Parallel and Distributed Programming Models in Heterogeneous Systems", *IEEE Transactions On Parallel and Distributed Systems,* Vol. 14, No. 11, November 2003.

[2] Charles Zhang and Hans-Arno Jacobsen, "Refactoring Middleware with Aspects", *IEEE Transactions On Parallel and Distributed Systems,* pp1058-1073, Vol. 14, No. 11, November 2003.

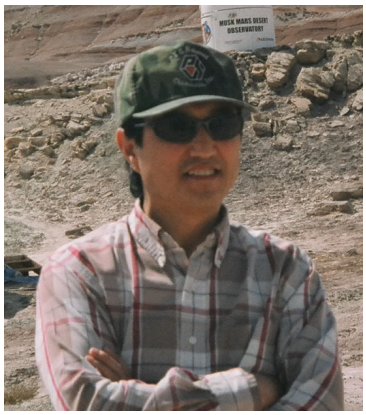
[3] L. Garces-Erice and E.W. Biersack and P. Felber and K.W. Ross and G. Urvoy-Keller. "Hierarchical Peer-to-Peer Systems", *Parallel Processing Letters*, Volume 13, Issue 4, December 2003.

[4] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec,"The Many Faces of Publish/Subscribe", *ACM Computing Surveys*, Volume 35, Issue 2, pp 114-131, June 2003.

[5] Victor Fay-Wolfe, Lisa C. DiPippo, Gregory Cooper, Russell Johnston, Peter Kortmann, and Bhavani Thuraisingham, "Real_Time CORBA", *IEEE Transactions On Parallel and Distributed Systems,* Vol. 11, No. 10, October 2000.

[6] Wenbing Zhao, Louise E. Moser, and P. Michael Melliar-Smith, "Unification of Transactions and Replication in Three-Tier Architectures Based on CORBA", IEEE Transactions on Dependable and Secure Computing, pp 14- 23, Vol. 2, No. 1, January-March 2005.

BIOGRAPHY



Charles Lee, *employed by SAIC, is a Technical Lead on Mobile Agents project at NASA Ames Research Center. He holds a Ph.D. in systems engineering and computer science from Oakland University, in Rochester, Michigan. Completed research projects includes robust GPS switchboard on-demand services that provide GPS information with awareness of*

loss and the ability to regain wireless network connections, and a store and forward architecture to maintain data continuity in the event of network connection loss. Other work includes join development of custom software to provide access to avionics data for Advanced Diagnostics System (ADS) and Integrated System Health Management (ISHM) applications, and collection and organization of International Space Station (ISS) data sets by fault scenario, along with liaison with ADS, ISHM developers and users in the design of data interfaces, user interfaces and tools relevant to ADS, ISHM on ISS. He developed the first version of Caution and Warning cube visualization software that handles the command and data handling events for fault detection, and then was evolved to Strider application.

Thom Stone *is a Senior Computer Scientist with Computer Sciences Corporation. He is attached to the NASA NREN (NASA Research and Engineering Network) project at Ames Research Center (ARC). Stone has been at NASA ARC employed by various contractors since 1989. He was an engineer with the NASA Science Internet (NSI) project office where he led the project that bought reliable Internet connections to remote locations including US bases in Antarctica, McMurdo Station and Amundson Scott South Pole Station. He was principal engineer for communications for the NASA Search for Extraterrestrial Intelligence (SETI) project and was a senior engineer for the Space Station Biological Research Project (SSBRP). Before his involvement with NASA, Stone was employed in the computer and communications industry and taught telecommunications at the undergraduate level.*

Richard Alena *is a computer engineer and the group lead for the Intelligent Mobile Technologies (IMT) Lab and the Mobile Exploration System (MEX) testbed at NASA Ames Research Center. The IMT team integrates mobile hardware and software components into unique systems capable of extending human performance aboard spacecraft during flight and payload operations. He was principal investigator for the Wireless Network Experiment flown aboard Shuttle and Mir, technology later adopted by the International Space Station Program. Alena spent four summers in the Canadian Arctic developing mobile technologies for human planetary exploration. He is a co-investigator on the Mobile Agents project, which is conducting field simulations in the American southwest. He has a MSEE&CS from University of California, Berkeley.*

John Ossenfort *is a network/ systems administrator at the NASA Ames Research Center, specializing in wireless communications. He has been acting systems administrator for the IMT Lab and the MEX testbed for the past three years and has accompanied the Mobile Agents team on four field simulations in the Arizona/Utah desert, assisting in all aspects of wireless network design, deployment, troubleshooting and maintenance. He is currently working on a wide array of projects, from Martian subsurface drilling simulations to Integrated Systems Health Management for the International Space Station. John has a dual BA degree in Anthropology and East Asian Studies from Washington University in St. Louis. He currently resides in Los Gatos, California.*

Ed Walker *is a hardware and networking specialist with the Intelligent Mobile Technologies (IMT) team at NASA Ames Research Center. He is a member of the team developing and testing the capabilities of the Mobile EXploration (MEX) testbed. The MEX System is a model for human planetary exploration that incorporates rugged computing, long-range wireless communication and mobility in support of planetary explorers. He graduated Foothill College in Los Altos, California with AS degrees in Data Communication & Network Management as well as Enterprise Networking.*

Hugo A. Notario *has been involved in software and hardware architecture since 1994. He has a BA in Industrial Engineering with an option of electronics in 1988 and 1997 earned an AA in Computer Service Technology. In the past 5 years I have earned several technical degrees in software and networking. I am currently working on my second bachelor in Computer Science. He joined NASA/Ames around two years ago in which he has been involved in Sensor Data Distribution in Planetary Exploration.*